Towards Automated Detection of Mobile Usability Issues

Daniel Bader
Technische Universität München
Munich, Germany
baderd@cs.tum.edu

Dennis Pagano
Technische Universität München
Munich, Germany
pagano@cs.tum.edu

Abstract: While evaluating the usability of mobile applications in the field has proven to lead to better results than in laboratory settings, in practice it is still not carried out after deployment – typically due to the required resources. In this paper we demonstrate a lightweight automated method for revealing specific usability issues of mobile applications in the field. Based on application usage data, we derive a simple heuristic which detects low discoverability by analyzing view transitions of mobile applications at runtime. We show the applicability and feasibility of our approach in a user study with a real application. Our results are promising and call for further research.

1 Introduction

Usability is a major selling point of today's software and particularly important for mobile applications which have to deal with relatively small screen estates [Hua09]. As a consequence, usability evaluation has become a mainstream activity over the last decades [CWK10]. While typical usability evaluations are carried out with a subset of all users in laboratory conditions, research has shown that evaluating usability in the field leads to significantly better results, including the identification of more issues and additional issue types [NOBP+06]. This is especially true for mobile applications, which are used in changing contexts that are often unknown before their deployment. In addition, application distribution platforms make mobile applications available to the general public and lead to more heterogeneous user audiences with different behavior and mental models. But even though it provides valuable results, usability evaluation still does not play a substantial role after deployment in practice [CKW+11], mainly because evaluating usability in the field is typically more complex and requires more resources [NOBP+06, RRH00]. As a consequence, research started to investigate to which extent usability evaluations can be automated [IH01], for instance by collecting usability relevant data like user interaction traces and automatizing its analysis [HR00]. Building on these foundations, we aim at developing lightweight and cost-effective methods for evaluating the usability of mobile applications in the field.

In this paper, we provide a proof of concept for automatically detecting *low discoverability issues* – a specific usability issue type which occurs whenever a user interface does not communicate clearly that and how the user can interact with a particular element. As a result, low discoverable user interface elements are often not found, and might even hinder

users to access particular views. Consequently, Nielsen considers low discoverability as one of the main challenges for user interfaces on touch-based mobile devices [BN].

The contribution of this paper is twofold. First, it describes a simple heuristic to detect low discoverability issues, which is derived from exploring real application usage data. Second, it shows the practical applicability of this mobile usability heuristic in the form of a lightweight framework. The framework can be integrated into mobile applications and allows for automated usability testing during application runtime.

The remainder of the paper is structured as follows. Section 2 introduces our research methodology. In the following three sections, we explore mobile application usage data (Section 3) to derive a heuristic for the detection of low discoverability issues (Section 4), and evaluate the heuristic in a user study (Section 5). Section 6 discusses the implications and limitations of our findings, while Section 7 summarizes related work. Finally, Section 8 concludes the paper and sketches our future plans.

2 Research Setting

We first summarize the questions that drive our research. Then we describe the method we used to collect and analyze application usage data and to evaluate the derived mobile usability heuristic.

2.1 Research Questions

Our goal is to investigate if low discoverability issues can be identified automatically by analyzing users' interactions. With this approach, we aim at providing groundwork towards a lightweight and cost-effective framework for mobile usability testing. Specifically, we focus on the following three research questions:

- **RQ 1: Issue indicators.** Does low discoverability appear in user interaction traces?
- **RQ 2: Detection heuristic.** Are there specific user interaction patterns which indicate the presence of a low discoverability issue?
- **RQ 3: Feasibility and applicability.** Can low discoverability issues be detected at runtime by a software framework?

2.2 Research Method

As depicted in Figure 1, our research methods consisted of three phases: a usage data collection phase, a heuristic construction phase, and an evaluation phase.

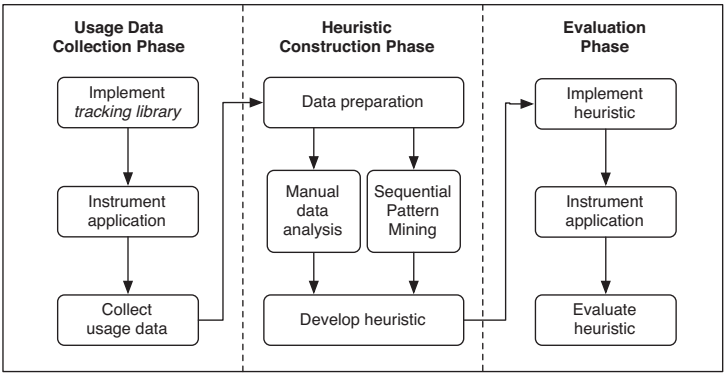

Figure 1: Research method.

In the initial *usage data collection phase*, we recorded users' interactions with a test application in order to build a corpus for further analysis. To this end, we first developed a generic tracking library which collects information about user interactions at runtime and that can be embedded into arbitrary mobile applications running on Apple's iOS[1] platform. The library records the time at which the user switches to a different view in the application together with an identifier for the new view, and logs activation and deactivation of the application. We then selected an existing mobile application for usage data collection. Apart from the obvious requirement that the application had to run on the iOS platform, we also needed access to its source code to embed the tracking library. Moreover, we had to select an application with a non-trivial navigational path, meaning that it had to contain multiple individual views and that using the application also required visiting them. Finally, we instrumented the selected application with the tracking library, and performed the actual data collection study. To this end, we asked subjects with different prior knowledge of the application to perform a sequence of tasks with it, and recorded the emerging usage data together with the ground truth about occurring low discoverability issues.

The subsequent *heuristic construction phase* consisted of four steps. After preparing the collected usage data for further analysis, we examined the obtained user interaction traces for regularities both manually and by applying a sequential pattern mining algorithm. We then used our findings to derive a heuristic classifier for low discoverability issues which works on user interaction traces.

In the *evaluation phase* we assessed the feasibility of automatically detecting low discoverability issues at runtime. As proof of concept, we implemented the detection heuristic as a software framework that can be integrated into arbitrary iOS applications. The framework constantly analyzes the occurring view transitions. Whenever the included heuristic detects a low discoverability issue, the framework notifies the user and asks if she agrees. Finally, we conducted a user study to investigate the quality of the detection heuristic. For

[1] `http://www.apple.com/ios`

this purpose, we let a different set of test users work with the instrumented test application and investigated the results in terms of true and false positives and negatives.

3 Usage Data Collection

3.1 Setup

To record how users interact with the test application we developed a *tracking library* that can be embedded into arbitrary iOS applications. The tracking library collects information about how a user navigates through the views of the application. It writes a view *presentation log* that contains a sequence of timestamped *view presentation events*. View presentation events are tracked by invoking a method whenever a view is presented to the user. Likewise, leaving and re-entering the application at runtime are encoded as special view presentation events. The data logging method requires access to the test applications source code in order to place logging statements at the relevant positions. It is self contained and works on standalone iOS devices without external connections, what enables the collection of usability data in real usage situations and locations where it is difficult to observe users directly. This benefits the results' validity because users behave more naturally when the evaluation is conducted in a familiar environment [PRS07].

We used the tracking library to collect usage data in single-participant sessions. In each session a test user interacted with a test application that was instrumented using the tracking library. All test users were asked to complete a sequence of 13 tasks. No further instructions were given during the session except when a user was stuck for more than one minute. During the sessions the test users were monitored in two ways: first, by an observer who took notes about the user's progress and second, by the integrated usability tracking library. In addition, all test users filled out a questionnaire after completing the tasks. The questionnaire contained a section for each of the 26 application views and required the test users to rate each view in the four dimensions discoverability, accidental activation, importance, and user confusion on a 5-point Likert scale (Figure 2).

After a session the recorded view presentation logs were drawn from the mobile devices for further processing. We then transformed the collected view presentation sequence into a sequence of *view visitation events*, each of which contained the name of the visited view and the *retention time*, indicating the number of seconds a user stayed in the view before leaving it. We performed this transformation in two steps. First, we corrected all timestamps by removing any time periods where the application was in the background. Second, we generated one view visitation event for each of the presented views by computing the retention times.

We used the iPhone application *MoID* as test application during usage data collection. MoID is an electronic replacement for business cards available in the iOS App Store [MoI]. The application fit the requirements from Section 2.2 because it runs on the iOS platform and its source code was accessible to us. Moreover, MoID is a sufficiently complex application consisting of 26 individual views.

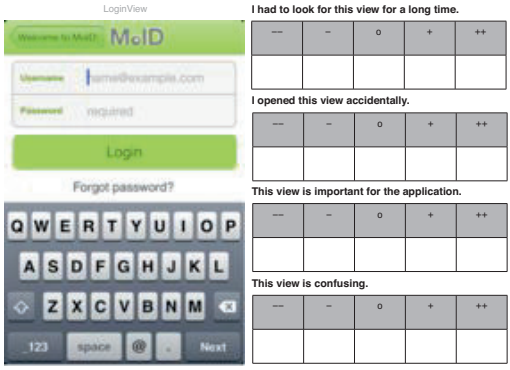

Figure 2: Sample section from data collection questionnaire.

Table 1: Overview of collected usage data.

# test users	6
smartphone owners	50%
average session length	18:09 minutes
# log file entries	581
time spent in scanning phases	722 s (12.5%)
time spent in working phases	5064 s (87.5%)

3.2 Results

An overview of the collected data is shown in Table 1. We conducted individual testing sessions with a group of 6 test subjects (3 female, 3 male). Their average previous knowledge of the test application was 2.2 on a subjective scale between 1 to 5 (1 indicating no previous knowledge, 5 indicating very high knowledge). Half of the test users owned smart phones, 1 test user owned an iPhone. During the sessions, 581 view presentation events were recorded by the usability tracking library. Each test user visited at least 16 of the 26 views of the application (61% minimum coverage) while the combined coverage of all test users was 23 out of 26 views (88% coverage).

To gain first insight into how the test users navigated through the application we analyzed the distribution of the retention times. As shown in Figure 3, there are many view visitations with short retention times and few visitations with long retention times. This observation suggests that retention times follow a power-law distribution [New05].

Next, we compared the performance of novice users to that of experienced users with previous experience with the test application. Two differences between their performances became apparent: First, novice users visited many views in order to complete the given tasks. Experienced users on the other hand exhibited almost no searching behavior, presumably since they knew what to do to solve the given tasks. Second, novice users required much more time in total to complete the given tasks than experienced users.

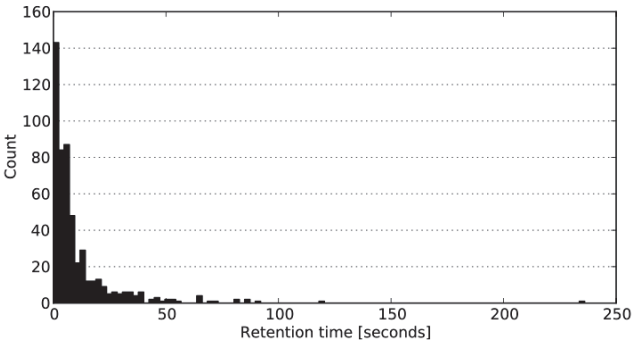

Figure 3: Ordered histogram of users' retention times.

After an initial analysis of the collected view visitation sequences we noticed that there was no connection between the observed difficulty a user had with a view and how she rated that view in the questionnaire. For example, some test users struggled with finding a view during the session but later indicated that they had no issue with the respective view in the application. We therefore decided to concentrate more on the collected observer notes for further analysis.

4 Heuristic Construction

Manual analysis of the view visitation sequences of each user revealed two kinds of behaviors that we call *scanning* and *working* phases. During *working* phases users stayed on one view for a long time – often longer than 10 seconds. These phases coincided with users working on a particular task, for example, entering information or reading descriptions. In *scanning* phases users switched rapidly between views and thus produced retention times of less than 6 seconds. Such phases appeared most often when users were looking for a particular view or function in the application in order to continue with their task.

We compared the observer notes with the view visitation sequences and investigated tasks that were difficult for the test users. We found that whenever a user had problems finding a particular view or function, the corresponding sequence typically included an overly long and misguided scanning phase. We called these misguided and unsuccessful scanning phases *problematic* scanning phases. Problematic scanning phases are characterized by three attributes in our data set. First, their view visitation events have retention times of less than 6 seconds. Second, they consist of sequences of at least 6 view visitations. Third, they contain loops, i.e., users visited at least one view multiple times during a problematic scanning phase. Figure 4 depicts how scanning phases and problematic scanning phases are identified in a user's view visitation sequence.

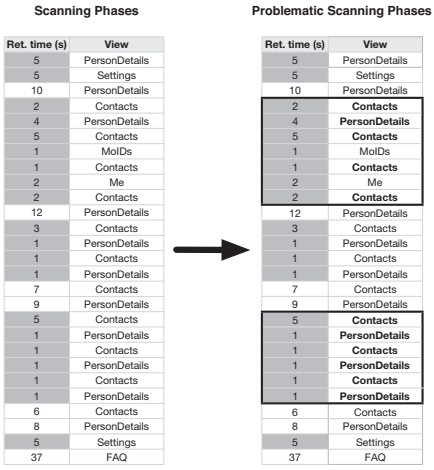

Figure 4: Example of problematic scanning phases in a view visitation sequence.

Our observer notes for each session indicated that problematic scanning phases are correlated with the occurrence of low discoverability issues. Several users failed at tasks because they missed important user interface controls in the application. Whenever a low discoverability issue occurred, users typically started to switch quickly between the views of the application. This searching behavior continued until the user could locate the user interface control or view she was looking for. We therefore hypothesized that problematic scanning phases are linked to the occurrence of low discoverability issues.

Since manual identification of problematic scanning phases is impractical for analyzing large data sets with many users in practice, we investigated if we could obtain the same results using automated techniques. To this end, we replicated the manual identification of problematic scanning phases using a sequential pattern mining algorithm by Zaki [Zak01]. For each view presentation log we generated all sequential patterns up to a maximum length of 6 items with a maximum gap value of 5. This step generated 629 patterns but this figure also includes subsequences, i.e. if the pattern list included the pattern (X, Y, Z) then it would also contain the subsequences (X, Y) and (X). After removing the superfluous subsequences we received a list of scanning phases in the input data. To identify problematic scanning phases we had to filter the resulting list of scanning phases by removing all sequences which contained no loops. Eventually, we obtained the same problematic scanning phases which we had identified manually.

We found that working and scanning phases are present in the recorded view visitation logs, assuming that problematic scanning phases consist of a sequence of view visitations with low retention times and loops. To facilitate further analysis and allow for practical use, in the following we describe a heuristic which detects these phases in sequences of view visitations. We begin by introducing the necessary terminology:

- A *view visitation* is a pair $(view, retentionTime)$.
- The *retention time* is the time in seconds the user stayed on a view. If a user enters view *A*, stays there for 23 seconds, and then navigates to view *B*, view *A*'s retention time adds to 23 seconds. This is expressed as the view transition $(A, 23)$.
- A *view visitation sequence* is an ordered list $(v_1, \ldots, v_n)$ of *view visitations*. The sequence is ordered by the natural order of events, e.g. if the view visitations A, B, and C occur one after another they are represented as the sequence (A, B, C).
- The *view visitation history* $H = (v_1, \ldots, v_N)$ is a special view visitation sequence that contains the last N view visitations. The view visitation history can be seen as a circular buffer and it is in fact implemented as such in our prototype.

We define the heuristic $ld\,(H)$ that detects low discoverability issues from a view visitation history. The occurrence of a low discoverability issue is indicated by low retention times and one or more loops within the view visitation history:

$$ld(H) := \begin{cases} \textbf{true} & \Leftrightarrow lowR\,(H) \wedge loop\,(H) \\ \textbf{false} & otherwise \end{cases} \tag{1}$$

The heuristic $ld\,(H)$ takes the view visitation history H as input and outputs a truth value that indicates if the heuristic was *triggered*, i.e. if a low discoverability usability issue is present in the input data. In the definition of ld we used two helper predicates, $lowR\,(H)$ and $loop\,(H)$. The $lowR\,(H)$ helper predicate indicates if all view visitations in the view visitation history H have retention times below a threshold value α:

$$lowR\,(H) := \begin{cases} \textbf{true} & \Leftrightarrow (\forall a_i \epsilon H : retentionTime\,(a_i) < \alpha) \\ \textbf{false} & otherwise \end{cases} \tag{2}$$

The predicate $loop\,(H)$ indicates if a view appears more than once in the view visitation history H, meaning that the user has visited a view at least twice in the last N view transitions:

$$loop(H) := \begin{cases} \textbf{true} & \Leftrightarrow (\exists a_j, a_k \epsilon H : a_j \neq a_k \wedge view\,(a_j) = view\,(a_k)) \\ \textbf{false} & otherwise \end{cases} \tag{3}$$

In our data set a history size of $N = 6$ view visitations was sufficient to distinguish problematic scanning phases from non-problematic. Likewise, a low retention time threshold of $\alpha = 6$ seconds was the most helpful when determining whether a given sequence of view visitations was a scanning or a working phase.

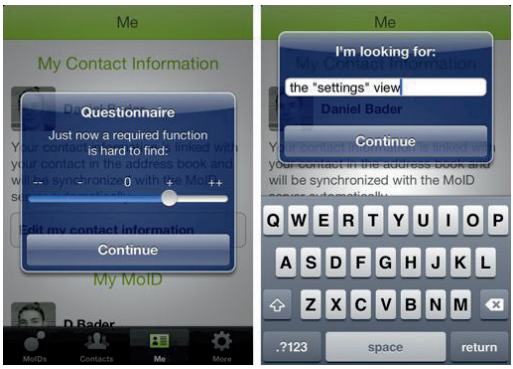

Figure 5: Screenshot of the automatic feedback survey.

5 Evaluation

We tested the heuristic in a first evaluation study with the same application and the same tasks as in the initial usage data collection study (Section 3), but carried out by a different set of users. As preparation, we implemented the heuristic as a library and included it in the test application binary. The instrumented test application analyzes the view visitation history after each view transition by the user and checks it for the occurrence of low discoverability issues. Whenever the heuristic is triggered, the application displays an *automatic feedback survey* dialog (shown in Figure 5), asking the user to rate if she currently struggles with finding a specific application element. To continue working with the application, the user has to answer the dialog by specifying her level of agreement on a 5-point Likert scale. Upon agreement, she is additionally asked to briefly specify what she was looking for. The user's response along with a timestamp and the view transition history is logged on the mobile device.

Table 2 shows an overview of the evaluation data. The evaluation study was performed with 9 test users. The level of previous knowledge with the test application is slightly lower than in the data analysis study. The average session length decreased by 33% from 18 to 12 minutes.

After each session, we determined the number of true and false positives and negatives by analyzing the accumulated log file. A *true positive* occurs when the heuristic is triggered if a low discoverability issue is present. Likewise, a *false positive* occurs when the heuristic is triggered although no low discoverability issue has been observed. In contrast, a *true negative* occurs when the heuristic is not triggered but also no low discoverability issue is present. Similarly, a *false negative* occurs when the heuristic fails to trigger, although a low discoverability issue is present. In our setting, at most $\#view\ transitions - 6$ low discoverability issues may be detected per session, which allowed us to calculate the number of true negatives. We finally estimated the number of false negatives based on our observer notes and a paper-based questionnaire after the session.

Table 2: Evaluation data overview.

# test users	9
smartphone owners	67%
avg. session length	12 minutes
# heuristic triggers	13
avg. # heuristic triggers per person	1.4
# log file entries	729

Table 3: Confusion matrix of low discoverability heuristic.

		Ground truth	
		LD^+	LD^-
Prediction	LD^+	8	5
	LD^-	3	568

Figure 6 depicts the evaluation results. It shows two bars for each test user, representing the number of times the heuristic was triggered during the evaluation session (prediction) and the number of times the test user agreed with the presence of a low discoverability issue in the automatic feedback survey (ground-truth). We found that the heuristic was triggered at least once in 6 of the 9 sessions (66%). Out of the 6 corresponding participants, 3 agreed with all predictions, 2 agreed partially, while 1 participant disagreed completely, leading to an average precision of 64% for the heuristic. If we additionally include the 3 participants who never triggered the heuristic, we arrive at a precision of 76%.

The confusion matrix shown in Table 3 illustrates the obtained evaluation results.The high number of true negatives indicates that the heuristic was typically not triggered erroneously. Furthermore, the fact that there are more false positives than false negatives means that the heuristic is more likely to trigger accidentally than to miss a real issue.

We used the confusion matrix to compute additional performance metrics for the heuristic. To do so we evaluated the heuristic like a binary classifier that predicts whether or not a low discoverability issue occurred after each view transition. Using this method we arrive at a specificity of 0.99, a sensitivity (recall) of 0.73, and a precision (confidence) of 0.61. The high specificity indicates that positive predictions by the heuristic have a high probability of being correct. Additionally, we computed the Matthews Correlation Coefficient (MCC) and Cohen's Kappa Score for the confusion matrix. Both metrics are frequently used to judge the performance of binary classifiers. The MCC and Kappa scores are both 0.66, thus indicating that our approach performs better than a classifier that randomly guesses its results.

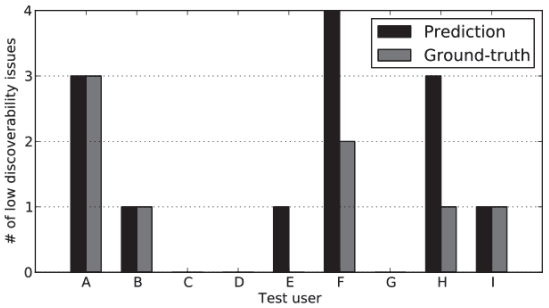

Figure 6: Evaluation results in terms of predicted issues and ground truth.

6 Discussion

6.1 Implications

Our findings represent a starting point towards testing and maintaining the usability of mobile applications in an automated way. We have shown that specific usability issues can be detected by a lightweight analysis of user interaction traces. By inspecting how users navigate through a mobile application we were able to identify patterns which might indicate low discoverability issues.

On the one hand, our findings provide foundational work for further research which should explore the potential of the proposed approach and exploit its benefits. We see three main directions. First, the applicability of user interaction trace analysis to other usability issues should be studied. To this end, researchers have to investigate which information is necessary to identify and classify additional usability issues and their respective causes. Second, researchers should collect and analyze user interaction traces from multiple users, in order to explore if specific usability issue patterns can be identified by aggregating this data. The results might provide means for an individual adjustment of usability issue detection heuristics or help to identify previously unknown issue patterns. To this end, heuristics like the one proposed in this paper could be integrated into existing software maintenance frameworks such as FastFix [PJB+12], or be made available as service working across multiple applications. Third, an automated detection of usability issues allows for a reaction at runtime. Apart from collecting data to improve usability, researchers may therefore explore ways to react *individually* to detected issues. Adaptive user interfaces might, for instance, increase button sizes if users continuously fail to select them. Moreover, contextual help might appear if and only if users are confused by a specific user interface.

On the other hand, the few requirements of our approach fit well with the restrictions of mobile devices, such as limited processing capability and power [ZA05], and ensure its practical feasibility. We have implemented the presented detection heuristic as a lightweight, application independent library, thus allowing practitioners to collect post-deployment usability data for real-world applications with little impact on battery life and performance. As a first step, developers may simply collect statistics about the usability of their applications, try to identify problematic user interfaces, and improve them. In the long run, usability data gathered from many users has to be visualized and integrated with other development data and tools.

6.2 Limitations

As with any research methodology, our choice of research methods has limitations. We therefore discuss the three main factors which might have affected the soundness of our work, and illustrate how we tried to limit them.

First, the *construct validity* of our approach might be affected if the described heuristic does not measure the occurrence of low discoverability issues but rather a related concept

such as “user confusion”. To limit this threat, we derived our heuristic from triangulated data, i.e. we aligned both user interaction traces and qualitative data gathered from users with questionnaires.

Second, the *internal validity* of our approach might be affected as study participants might have provided answers which do not completely reflect the reality, because they knew the results would be published. While this threat can never be fully eliminated in studies with real participants, we addressed it by guaranteeing our participants complete anonymity.

Finally, the applicability of our findings has to be established carefully. The main limitation to their *external validity* results from the fact that we have constructed and evaluated the proposed heuristic based on only a single application, and from the relatively small sample of test subjects. As a result, the heuristic may therefore be tied to usability issues that appear in this specific application. On the one hand, the proposed heuristic includes two parameters which allow for further adjustment to other applications and user behavior. On the other hand, our study was not designed to be largely generalizable. Rather than predicting the heuristic’s performance for all possible scenarios, its main idea is to explore the feasibility of automated usability issue detection by analyzing user interaction traces, and therefore to provide a proof of concept. Further research is necessary to validate our findings for the general case. Therefore we make the detection library available to other researchers to enable the replication of our study[2].

7 Related Work

Hilbert and Redmiles [HR00] give a comprehensive overview of techniques to extract usability-related information from user interface events of conventional desktop applications. In contrast, research on automated usability evaluation of *mobile* applications is still in an early stage. We therefore focus the related work discussion additionally on web applications, as we found this area to be the most mature. For instance, Vargas et al. [VWdR10] aim at automatically detecting usability issues in web applications. Similar to our approach, the authors propose to match user interaction sequences from web server log files against heuristics which have been specified a priori. Atterer and Schmidt [AS07] additionally show how to extend user interaction logging to include client side browser events with an AJAX based HTTP proxy. In general, conventional web applications differ from mobile applications mainly in two aspects: their user interface design and the user interaction methods. As a result, Vargas et al. perform their analysis on more data such as mouse movements, keystrokes, and accessed links. Nevertheless, our findings confirm that a similar approach is also feasible for mobile applications, and that usability issues can be detected with less interaction data available.

Most research on usability evaluation of mobile applications is concerned with supporting early design and prototyping of mobile user interfaces (e.g. [ABWD08, dSC09]). In contrast, Patern et al. [PRS07] describe a framework for the remote evaluation of applications

[2] `http://dbader.org/me13-paper`

on the Microsoft Windows CE[3] platform. Their approach is based on a task model, specified upfront by developers, which defines how users should work with the application. The framework logs users' behavior while an external software analyzes the collected data and identifies deviant user behavior, which may hint at usability issues. While our approach is conceptually similar, it works in real-time without an external analysis and without setting up an application dependent task model. As a result, our framework is more lightweight and requires less setup effort, but on the other hand might provide less flexibility. Google Analytics for Mobile [Goo11] is an extension of the Google Analytics framework for web pages, which allows developers to track user interactions within mobile applications for the Google Android and Apple iOS platforms. To use the framework, developers have to trigger specific events by calling a function in the Analytics library. A list of the triggered events is then periodically sent to a remote server for further analysis. Moreover, the framework includes a web application which summarizes and visualizes the collected data. In contrast to our work, Google Analytics does not automatically detect usability issues but instead analyzes the overall performance in the context of all users.

8 Conclusions

Evaluating the usability of mobile applications in the field has proven to lead to better results than in laboratory settings. However, in practice it is typically not carried out after the software is deployed, because the required resources are higher and such an evaluation has a higher degree of complexity. The goal of our research is to facilitate post-deployment usability testing of mobile applications by developing lightweight and cost-effective evaluation methods. In this paper we demonstrated how to detect low discoverability issues by a simple analysis of users' view transitions at runtime. We showed the applicability and feasibility of our approach in a user study with a real application. We found that our lightweight framework for automated usability testing has the potential to be integrated into other existing mobile applications.

Our results represent a starting point towards an automated evaluation framework of mobile usability in the field. In addition to this proof of concept, future research has to investigate how to detect and mitigate various different usability issues and how to benefit from usability data gathered from many users.

References

[ABWD08] F. Au, S. Baker, I. Warren, and G. Dobbie. Automated usability testing framework. *Proceedings of the 9th conference on Australasian user interface*, pages 55–64, 2008.

[AS07] R. Atterer and A. Schmidt. Tracking the Interaction of Users with AJAX Applications for Usability Testing. In *Proceedings of CHI '07*, pages 1347–1350, San Jose, CA, USA, 2007. ACM.

[3] `http://msdn.microsoft.com/en-ph/embedded`

[BN] R. Budiu and J. Nielsen. Usability of iPad Apps and Websites. http://www.nngroup.com/reports/mobile/ipad/ipad-usability_1st-edition.pdf. Retrieved October 10, 2011.

[CKW+11] P. K. Chilana, A. J. Ko, J. O. Wobbrock, T. Grossman, and G. Fitzmaurice. Post-Deployment Usability: A Survey of Current Practices. In *Proceedings of CHI '11*, pages 2243–2246, Vancouver, BC, Canada, 2011. ACM.

[CWK10] P. K. Chilana, J. O. Wobbrock, and A. J. Ko. Understanding usability practices in complex domains. *Proceedings of CHI '10*, page 2337, 2010.

[dSC09] M. de Sá and L. Carriço. An Evaluation Framework for Mobile User Interfaces. *Human-Computer Interaction–INTERACT 2009*, 2009.

[Goo11] Google Inc. Developer's Guide - Google Analytics for Mobile. http://code.google.com/mobile/analytics/docs, February 2011.

[HR00] D. M. Hilbert and D. F. Redmiles. Extracting usability information from user interface events. *ACM Computing Surveys*, 32(4):384–421, December 2000.

[Hua09] K. Y. Huang. Challenges in human-computer interaction design for mobile devices. In *Proceedings of the World Congress on Engineering and Computer Science*, San Francisco, CA, USA, 2009.

[IH01] M. Ivory and M. Hearst. The state of the art in automating usability evaluation of user interfaces. *ACM Computing Surveys*, 33(4), December 2001.

[MoI] MoID GmbH. Homepage. http://www.moid.de. Retrieved October 10, 2011.

[New05] M. E. J. Newman. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, 46:323–351, 2005.

[NOBP+06] C. M. Nielsen, M. Overgaard, M. Bach Pedersen, J. Stage, and S. Stenild. It's Worth the Hassle! The Added Value of Evaluating the Usability of Mobile Systems in the Field. In *Proceedings of NordiCHI'06*, pages 14–18. ACM Press, 2006.

[PJB+12] D. Pagano, M. A. Juan, A. Bagnato, T. Roehm, B. Bruegge, and W. Maalej. FastFix: Monitoring Control for Remote Software Maintenance. In *Proceedings of ICSE'12*, pages 1437–1438, Zurich, Switzerland, 2012. IEEE.

[PRS07] F. Paternò, A. Russino, and C. Santoro. Remote Evaluation of Mobile Applications. In *Proceedings of TAMODIA'07*, pages 155–169, 2007.

[RRH00] S. Rosenbaum, J. A. Rohn, and J. Humburg. A Toolkit for Strategic Usability: Results from Workshops, Panels, and Surveys. In *CHI'00*, pages 337–344. ACM, 2000.

[VWdR10] A. Vargas, H. Weffers, and H. V. da Rocha. A Method for Remote and Semi-Automatic Usability Evaluation of Web-based Applications Through Users Behavior Analysis. *Measuring Behavior*, pages 1–5, August 2010.

[ZA05] D. Zhang and B. Adipat. Challenges, Methodologies, and Issues in the Usability Testing of Mobile Applications. *Human-Computer Interaction*, 18(3):293–308, 2005.

[Zak01] M.J. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning*, 42(1):31–60, 2001.